

Compile Plugins for Debugging

Plugin Debug Configuration inline with Opencpn

Why?

The reason we want to do this is to find bugs in a plugin!!!

External Standalone Plugins which need debugging generally should be compiled within an existing operational OpenCPN Debug environment in order to be debugged and have the code fixed. To do this you first need to have a working [Windows Development Environment for Compiling using MS Visual Studio 2013](#).

Finding Errors & Fixing them

We need the errors in the form of a filename and which line(s) specifically, and preferably a copy of the "call stack" The next step is to actually fix the problem and test again. All of this is done easiest in a single development environment as described below.

Standalone Plugin Compilation for Release.

If you have a plugin that is mature and just needs to be compiled for windows release and NSIS packaged then compiling in the [standalone configuration](#) is generally easiest and quickest.

Standalone Plugin Compilation for Debug. It is perfectly reasonable to compile a standalone plugin for debugging and then copy the dll and data to the source\build\plugin directory and then run cmake -T v120_xp .. from source\build, then open MS Visual Studio, build and then *Debug > Start Debugging*.

Conceptual: How do we configure for Plugin Debugging?

External Plugins which need debugging should be setup in the same or similar structure to the way the 4 Internal Plugins are setup. See the bottom of [Setup Batch Files](#) for a 4 Internal Plugins Table which shows source and destination directories.

Example of a "Source" Directory under github:

```
C:\users\[username]\Documents\GitHub\obeta\**source**
source\plugins          <-- Git clone https://username/name_pi into
here.
source\buildwin         <-- name_pi binaries & special dlls, exe put
here.
source\build            <-- cmake commands are executed from this
```

```
directory.  
source\build\plugins      <-- copy name_pi.dll from the directory below  
to here.  
source\build\plugins\debug <-- copy compiled debug name_pi.dll from here  
source\build\plugins\release <-- copy compiled release name_pi.dll from here  
source\build\release      <-- copy buildwin binaries & dlls to here for  
running release.  
source\build\release\plugins <-- copy release name_pi.dll to here +  
source\plugins\name_pi\data\*.*.
```

The **source** might be named OpenCPN

Use the command prompt to git clone the plugin into source\plugins

```
cd source\plugins  
Git Clone https://github.com/username/name_pi  
Git Fetch
```

Now new source files and directories are found under source\plugins\name_pi.

This folder contains the data folder and other files necessary for cmake and compilation.
The git clone plugin files are located in the same directory as the sourced internal plugin files.

Assuming a fresh git clone & install, run

```
mkdir build  
cd build  
CMake -T v120_xp ..  
CMake --build . --config debug  
CMake --build . --config debug
```

A new Plugins folder is created build\plugins\name_pi directory with subfolders “debug” and “release”

```
**source**\build\**plugins**\name_pi\**DEBUG**\name_pi.DLL
```

So that MS VStudio Debug Build can run the four debug plugins we need to copy the DEBUG version of name_pi.dll up several levels to

```
**source**\build\**plugins**\name_pi.DLL
```

Also this name_pi.DLL needs its corresponding DATA directory from source\plugins\name_pi\DATA*.* copied into the same source\build\plugins directory.

A similar structure occurs for running a Release version, but the destination is source\build\release\plugins. However we are not interested in release versions here. We should follow the pattern for debugging.

1. Step by Step, For real, not difficult, if the plugin works

Which Plugins directory to use for git clone & debugging?

```
source\plugins
```

in MS VS Command Prompt, navigate to **source\plugins**

```
git clone https://github.com/[username]/[name]_pi
```

Example:

```
cd C:\Users\[username]\Documents\GitHub\obeta\OpenCPN\plugins
git clone https://github.com/seandepagnier/projections_pi
Cloning into 'vdr_pi'...
..
Unpacking objects: 100% (70/70), done.
Checking connectivity... done.
```

We now have a “source\plugins\projections_pi directory” with files and possibly a “data” directory
We have the necessary files and folder to CMake and compile.

1. If it exists, Copy the source\plugins\name_pi\data*. * to source\build\plugins\name_pi\data
2. Read name_pi README for necessary binaries & dlls.
3. Next download any required windows binaries & dlls listed in the plugin README
4. Expand binaries and place them in the source\buildwin directory.
5. Also copy them to source\build and source\build\release.

The projections_pi plugin has not DATA directory and requires no binaries.
CMake can now be run.

2. Run CMake & Compile

```
cd ..
cd build to source\build
cmake -T v120_xp ..
cmake --build . --target package --config release
cmake --build . --config debug
```

Projections_pi did complete successfully with 0 Errors and 0 Warnings.

However the 5 plugins I tried first all had errors I could not figure out. Errors... hmmm. These occur often due to the peculiarities of a plugins. Various techniques are required to get the plugin going.

NOTE: If source\plugins\name_pi and source\build\plugins\name_pi is removed you can start over with

cmake -T v120_xp .. etc and build successfully.

Once the plugin compiles successfully, with no errors, the compiled file need to be copied before testing and debugging the plugin.

3. Copy files and Setup for Debugging

```
Copy source\build\plugins\name_pi\debug\name_pi.DLL
TO source\build\plugins\name_pi.DLL
```

```
Copy source\plugins\name_pi\DATA\*.*
TO source\build\plugins\DATA
```

Alternative 1 - Create a link to the Debug DLL

TransmitterDan has suggested creating a link to point to the Debug DLL in [Cruiser Forums Development Environment Thread](#)

When building plugins and debugging you can create a link to the plugin thus you don't have to disturb the actual plugins folder.

1. Less confusion, everything stays the same standalone pi and opencpn debugging.
2. Control of link from one place.
3. Less copying of files/directories, thus keeping MSVS++ OpenCPN pristine, without all the pi stuff.
4. No need to copy the Debug dll and name_pi dir to source/build/plugins

Something like this should work:

```
cd OpenCPN\plugins
mklink /D vdr_pi c:\myplugins\vdr_pi
```

At this point Cmake will see **OpenCPN\plugins\vdr_pi** as a folder and it will configure the plugin for building. Git will ignore **OpenCPN\plugins\vdr_pi** however.

Then you should be able to build either from **\myplugins\vdr_pi** or you can build from within the overall OpenCPN build. The link is the same as being inline with OpenCPN except Git does see the link and shows it as an untracked folder which is ok as far as I can tell.

To get rid of the link but not the plugin folder itself:

```
cd OpenCPN\plugins
rmdir vdr_pi
```

Alternative 2 - Modify Plugin Manager

Modify PluginManager to point to the debug DLL location. This section should be modified with this added information, which adds the dll to the plugin manager and permits MSVC++ to find and run the debug plugin.

See Cruiser Forum Post:

<http://www.cruisersforum.com/forums/f134/navico-radar-plugin-v3-0-beta3-v3-907-released-190692.html#post2483769>

“Normally from MSVS++ Run OpenCPN in debug mode with the plugin built internally similar to Dashboard or Chartdownloader. To Debug externally compiled Debug DLL's the process is slightly different.

1. The plugin dll must be found by MSVC++ Opencpn, so pluginmanager must be adjusted to find it.
2. First compile the Standalone plugin for debug.
3. Locate the debug dll and copy it to where pluginmanager is set to find it.
4. Then run MSVC++ OpenCPN in debug mode.
5. Thus you must copy the debug.dll every time you build.

The easy way to avoid having to copy the Debug.dll every time is to make a few changes in pluginmanager.cpp which has pluginmanager take care of the dll copying, or to tell pluginmanager where to find the particular debug.dll.

```
bool PluginManager::LoadAllPlugIns(const wxString &plugin_dir, bool
load_enabled, bool b_enable_blackdialog)
{
    pConfig->SetPath( _T("/PlugIns/") );
    SetPluginOrder( pConfig->Read( _T("PluginOrder"), wxEmptyString ) );
    // Enable the compatibility dialogs if requested, and has not been
already done once.
    m_benable_blackdialog = b_enable_blackdialog &&
!m_benable_blackdialog_done;
    #if _DEBUG //Has for debug use
#ifdef __WXMSW__
    bool x;
    wxString f, volume, path, name, ext, mess;
    wxString destf, Dest =
_T("C:\\Builds\\OCPN\\OpenCPN\\build\\plugins\\Debug_pi\\");

    f =
wxFindFirstFile(_T("C:\\Builds\\OCPN\\OpenCPN\\build\\plugins\\dashboard_pi\\D
ebug\\*..*"), wxFILE);
    while (!f.empty())
    {
        wxFileName::SplitPath(f, &volume, &path, &name, &ext);
        destf = Dest;
```

```
        destf << name << _T(".") << ext;
        x = wxCopyFile(f, destf, true);
        f = wxFindNextFile();
    }
    mess = name;

    f =
wxFindFirstFile(_T("C:\\Builds\\OCPN\\OpenCPN\\build\\plugins\\wmm_pi\\Debug\\
*..*"), wxFILE);
    while (!f.empty())
    {
        wxFileName::SplitPath(f, &volume, &path, &name, &ext);
        destf = Dest;
        destf << name << _T(".") << ext;
        x = wxCopyFile(f, destf, true);
        f = wxFindNextFile();
    }
    mess << _T(" and ") << name;
    //x =
wxCopyFile(_T("C:\\Builds\\OCPN\\OpenCPN\\build\\plugins\\BR24radar_pi\\Debug\\
\\br24radar_pi.dll"),
    //
_T("C:\\Builds\\OCPN\\OpenCPN\\build\\plugins\\Debug_pi\\br24radar_pi.dll"),
true);
    if (0) { // Set 1 for BR24
        f =
wxFindFirstFile(_T("C:\\Builds\\OCPN\\OpenCPN\\build\\plugins\\BR24radar_pi\\D
ebug\\*..*"), wxFILE);
        while (!f.empty())
        {
            wxFileName::SplitPath(f, &volume, &path, &name, &ext);
            destf = Dest;
            destf << name << _T(".") << ext;
            x = wxCopyFile(f, destf, true);
            f = wxFindNextFile();
        }
        mess << _T(" and ") << name;
    }
    mess << _T(" copied to ") << Dest;
    //wxMessageBox(mess, _T("Debug - Plugin info"));

    m_plugin_location =
_T("C:\\Builds\\OCPN\\OpenCPN\\build\\plugins\\Debug_pi\\");
#endif
    #else
        m_plugin_location = plugin_dir;
    #endif
```

```
wxString msg(_T("PlugInManager searching for PlugIns in location "));  
msg += m_plugin_location;  
wxLogMessage(msg);
```

For example BR24radar_pi is not included for the moment due to comment of corresponding lines.

```
m_plugin_location =  
_T("C:\\Builds\\0CPN\\OpenCPN\\build\\plugins\\Debug_pi\\");
```

4. Start MS VStudio and start debugging.

1. Set the startup project to **Opencpn**
2. Pick **Build**". **It should just confirm everything is built. - Pick Debug > Start Debugging****

5. Plugins which work in Plugin Debug Configuration

- chartdldr_pi - internal opencpn
- dashboard_pi - internal opencpn
- grib_pi - internal opencpn
- wmm_pi - internal opencpn
- [Projections_pi Sean's](#) 2/26/2017 rgleason
- [Tactics_pi Tom Big_Speedy's](#) 2/26/2017 rgleason
- [Celestial Navigation Powl's](#) 2/27/2017 rgleason
- [Launcher_pi](#) 2/27/2017 rgleason

6. Plugins which do NOT work in Plugin Debug Configuration

- [VDR_pi version Mike - Rasbats](#) 2/26/2017 rgleason
- [DR_pi Mike - Rasbats](#) 2/26/2017 rgleason
- [Weatherfax_pi Sean's](#) 2/26/2017 rgleason

From:
<https://opencpn.org/wiki/dokuwiki/> - **OpenCPN Manuals**

Permanent link:
https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:developer_manual:developer_guide:compiling_windows:compiling_plugins_to_debug

Last update: **2017/09/29 13:43**

